The Prefabricated Website: Who Needs a Server Anyway?

Martin Holmes and Joseph Takeda

ABSTRACT

Project Endings, a collaboration between digital humanists and librarians, is devising principles (https://raw.githubusercontent.com/projectEndings/Endings/master/principles.txt) for building DH projects in ways that ensure that they remain viable, functional, and archivable into the distant future. Endings principles cover five components of project design:

- Data
- Products
- Processing
- Documentation
- Release Management

Previous *Endings* work has focused on Data and Products (Holmes 2017; Arneil & Holmes 2017) and diagnostic tools for monitoring project progress (Holmes & Takeda 2018 and 2019). This presentation will deal with the mechanics of Processing, focusing in particular on building large static sites which are resilient because they have no requirement for server-side technology at all. We will use the *Map of Early Modern London* (MoEML) project as a case study.

INDEX

Keywords: TEI and beyond: interactions, interchange, integrations and interoperability, TEI environments and infrastructures, TEI and publication, TEI and sustainability

1. Introduction

1 *Project Endings*, a collaboration between digital humanists and librarians, is devising principles (https://raw.githubusercontent.com/projectEndings/Endings/master/principles.txt) for building DH projects in ways that ensure that they remain viable, functional, and archivable into the distant future. Endings principles cover five components of project design:

- Data
- Products
- Processing
- Documentation
- Release Management

Previous *Endings* work has focused on Data and Products (Holmes 2017; Arneil & Holmes 2017) and diagnostic tools for monitoring project progress (Holmes & Takeda 2018 and 2019). This presentation will deal with the mechanics of Processing, focusing in particular on building large static sites which are resilient because they have no requirement for server-side technology at all.

2 We will use the *Map of Early Modern London* project (MoEML), one of the flagship Endings project, as a case study. Comprised of 2,000 TEI source files and 15,000 distinct entities, MoEML is a densely interlinked project that requires a sophisticated build process to create its website structure, the historical Agas Map interface, editions of primary source documents, various indexes and

gazetteers, and encyclopedia entries. As a flagship Endings project, MoEML has been a testbed for the scalability of the Endings principles. The MoEML site has 9,000 HTML files, 26,000 XML files, and over 5,000 images, and is around 2GB in size. Our presentation will cover a number of key techniques in the build process, including:

- Validation, validation, validation: XML, HTML, CSS, and TEI `<egXML>` example code is validated at every stage of the build process.
- Diagnostics to check all links and targets.
- Unique query-free URLs for all entities
- Generating the gazetteer, which includes every variant spelling of every placename.
- Pre-generating HTML fragments for AJAX retrieval for every entity.
- Processing and rationalizing `<rendition>` elements and `@style` attributes.
- Using document type taxonomies to build sitemaps and breadcrumb trails.
- Filtering of images to include only those actually used.

2. Why build a static site?

3 When in early 2019 the server which was hosting the tei-c.org website died, the *WordPress*-based main site disappeared from the Internet for a considerable time. Since *WordPress* is a database-dependent system, a single central database host is required to run it, and until a new server could be brought up hosting that database, the site remained unavailable. However, there was no such problem with the TEI Guidelines, which are statically-built and available in multiple locations as a matter of course.[1] A static site, however, can be replicated endlessly. All digital humanities projects will eventually end (Kirschenbaum 2009; Rockwell et al 2014), and their products will transition into minimally-curated archival hosting; static sites have much more chance of survival, availability and replication if they have no server dependencies.

4 The world of large-scale software development is also coming to similar conclusions for slightly different reasons. The JamStack initiative (JamStack.org) is also championing “modern web development architecture based on client-side JavaScript, reusable APIs, and prebuilt Markup,” in the interests of “better performance,” “cheaper, easier scaling,” “higher security,” and a “better developer experience.” Long-term archivability is not a primary goal of JamStack; instead, one of

their motivations is that a static site is far more easily deployed across Content Delivery Networks such as Akamai because it has no reliance on centralized back-end data sources such as a databases. Like JamStack, *Endings* advocates products based on pure client-side HTML5, CSS, and JavaScript.

3. The build process

5 *MoEML*'s static build process, which is managed by *Apache Ant*, takes the densely-encoded, tightly-linked XML collection created by our team and builds from it a massive, loosely-coupled collection of web resources comprising everything we can possibly imagine an end-user might want to see. Before we start, though, we first check whether the current state of the collection is worth building into anything at all. We validate (RELAX NG), we validate again (Schematron), and we check coherence (does every link point to something meaningful?), consistency (does everything conform to the encoding guidelines and the editorial guidelines?), and completeness (does everything mentioned actually exist?) via our diagnostic processes (Holmes & Takeda 2019). If a single file is invalid, or a single link is broken, or an id is used for two different entities, the build fails and the process stops. A website with errors is not worth building.

6 It is worth contrasting this rigorous suite of validation processes with the pre-*Endings* form of the *MoEML* website, which was based on an *eXist* XML database and to which project staff uploaded new and changed materials as they finished them (or thought they had finished them), when it occurred to them, or (sometimes) accidentally, while uploading other materials. Articles were "published" containing links to other articles not yet written, or person records not yet added to the personography. One encoder would add an item to the bibliography with a new id, while another happened to use the same id for a location; both documents would be uploaded, and, at best, links would break and, at worst, the processing would fail to handle the error and break the site. Such issues were not rampant, but they were omnipresent. We will have no more of that.

7 It should also be noted that these sorts of errors were not only caused by encoders; developers of the *MoEML* site had to be very careful that any code committed to the eXist server was error-free. Testing code changes required a parallel hosting environment, which was an additional burden to maintain, and to keep synchronized with the live site. But by checking the validity of our outputs in the static build, we also necessarily ensure that our processes work: if everything in the build is valid, then, at the very least, the code itself can be compiled and it produces valid documents.

Of course, this does not ensure that the code functions precisely the way we want it to, but, as we discuss later, the static build process gives the project time to ensure that the processes work as expected.

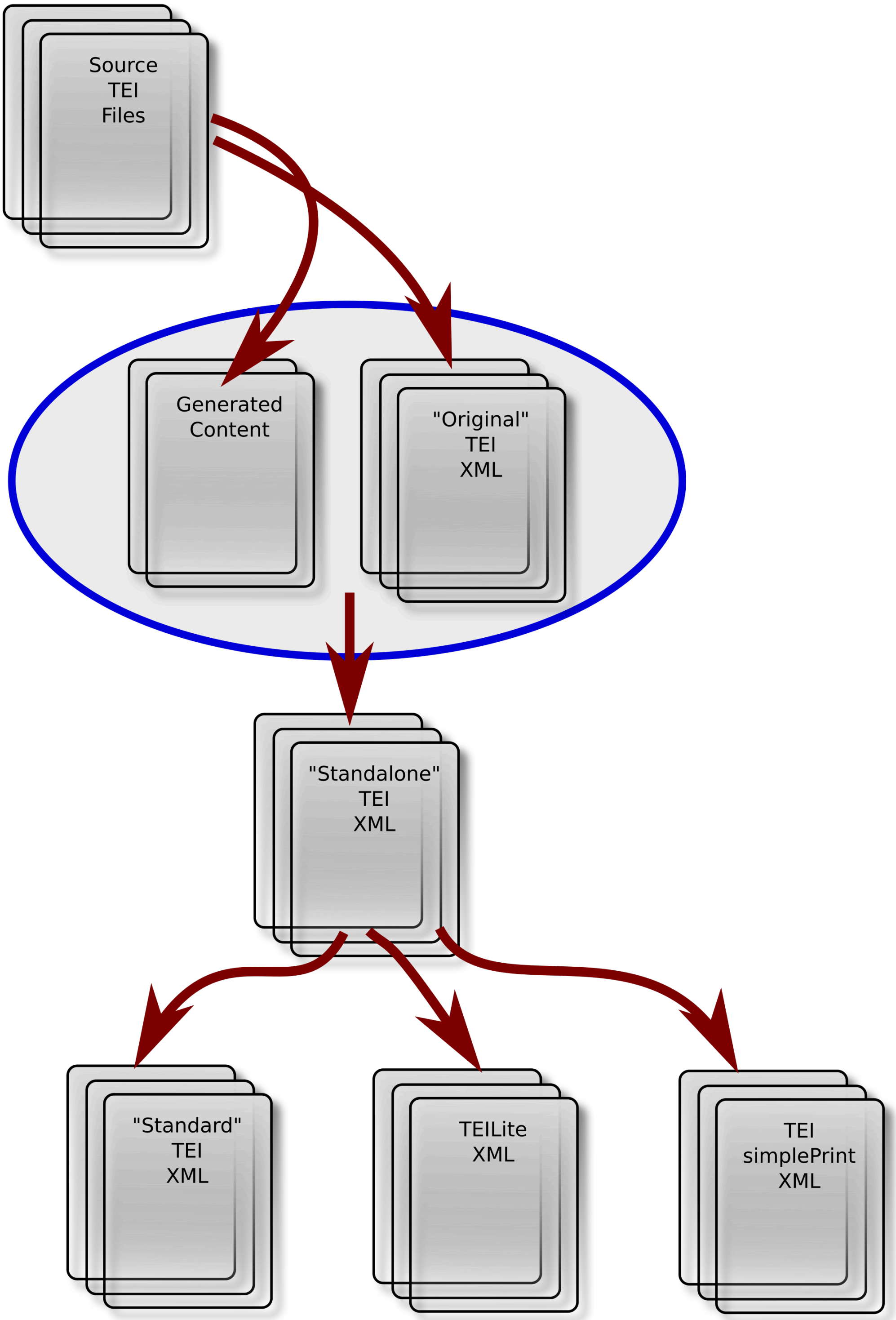
8 Assuming all the validation tests pass, the first stage in building the website is to make more TEI XML. Lots of it, in fact. We build five different versions of our XML collection (see figure 1). Holmes (2017b) provides a full description of the rationale behind this process, but the main justification is that we want to ensure that any future user who comes to our project looking for an XML document can likely find one that is tuned as closely as possible to their needs. We provide XML designed to best represent the praxis of our own project ("original" XML), XML designed to be less esoteric and that aligns more with standard TEI practices ("standard" XML), XML designed to be most amenable to generic external processors (TEI Lite, TEI simplePrint), and XML designed to be detached entirely from the rest of the collection, free of external links and dependencies ("standalone" XML). This is how we end up with 26,000 XML files, from a starting collection of only 2,000. As soon as each new version of the XML is created, you may easily guess what we do with it. We validate it. If any file fails validation, the build stops. This is also when we create a wealth of new files that did not exist before, including the project gazetteer and a range of compiled indexes and similar materials whose information is inherent to the original XML, but which can now be made explicit and tangible. More of this in the next section.

9 Finally, we begin to generate web products. A number of core principles govern the structure and organization of those products:

1. *Every entity* (location, person, bibliography entry, organization, article, etc.) has a *unique id*, and *every unique id gets its own individual page* on the site.
2. *No URL is ever abandoned.* If an id is changed or an entity is removed from the collection, a page is still generated, redirecting to the new version of the id, or to an explanation of what has happened. Linked Open Data requires stable identifiers, so we have a responsibility to maintain them indefinitely.
3. *Every page stands alone and complete in terms of content*; everything referenced in the body of the page (people, places, bibliography items etc.) is included into the page itself, so that if the page becomes detached from its context (if, for example, someone saves a local copy to use while disconnected from the internet), it will continue to work. This of course means that there is massive duplication of data across the site, but we don’t care. The entire site still comes in smaller than an HD movie.
4. *All pages live together in the same folder.* This makes for a very large folder, but it means that linking is trivial and reliable, and URLs are easily remembered and typed.

Finally, after the website content is generated, all its pages and associated CSS files are validated with the W3C’s VNU validator. As always, any invalid file causes the build to fail.

Figure 1. Generation of multiple TEI output formats.



4. Advantages: you can build anything

10 A major advantage of building the entire site offline is that we can run processes across the entire dataset to build any resource we like, no matter how time- or cycle-consuming it may be. A simple example is the "A-Z Index," which lists all 9,000 `@xml:ids` used in the project and provides information about the entity to which each id refers. This is an essential resource for *MoEML* encoders, who are often creating new globally unique `@xml:ids` for entities in the project; having a list of all `@xml:ids` not only prevents duplicates ids, but also ensures that encoders can check whether or not the person or place that they are creating already exists in the project. Just-in-time creation of the index is not a feasible option: the server-side construction of the page, which is nearly 10MB, would be slow, even on a powerful server. But, by creating this page ahead of time, the page downloads and renders reasonably rapidly. We also produce a plain-text list of all the ids for faster access and searching by encoders who may be on a slow internet connection.

11 Similarly, it would be impractical to generate the gazetteer of early modern London, which aggregates and groups the thousands of variant placenames across the project, from the source data on a live server. Before implementing the static build, this resource was manually compiled in a semi-automated process. Now we create these documents with the rest of the project, which ensures that these documents not only reflect the current state of the data, but are also completely valid HTML before they are published.

12 The offline build also allows us to take advantage of multi-step processes that would be very difficult to manage in a just-in-time rendering scenario. We make great use of the TEI `<rendition>/@selector` mechanism, which uses CSS selector syntax to specify TEI elements to which rendition descriptions apply when encoding presentational aspects of the input. In our build process, we use a two step process during the creation of the "standard" XML to resolve these CSS selectors. For each document that has a `//rendition[@selector]`, we create a temporary XSLT identity transform that converts the CSS selectors into XPath statements, which are then used as the `@match` value for a sequence of `<xsl:template>` elements. We run that transformation against the source document to create the "standard" version of the XML, adding `@renditions` that correspond with the predefined `<rendition>` in the header. In our standalone process, we then take all `@style` attributes on elements and abstract them into `@rendition` pointers to `<rendition>`

elements in the header. Then, in our HTML creation, all of the `<rendition>` elements are turned into class selectors in the header of the HTML and, accordingly, all `@rendition` attributes are converted into `@class` values.

5. Disadvantages

13 The primary disadvantage of this approach is of course that it involves deferred gratification. Builds take a long time, and they often fail to complete due to invalidities or other errors. It may be hours before an encoder or author can see the results of their work in the context of the built site, and this is particularly frustrating for those who are encoding primary source documents and trying to capture for reproduction rendering features of the original text.

14 However, patience is a virtue and cultivating it is no bad thing. Instant gratification is not a feature of scholarly discourse; compared with waiting for a journal article to be published, waiting a couple of hours to see the latest draft of your document in all its glory is scarcely a hardship. This virtue also extends to the discipline around the public release of complete new versions of a site. Rather than a “rolling release” publication model, where on any given day, the state of the site is inconsistent, incoherent, and unpredictable, a static build process demands a fixed released process, akin to the model of editions of a print text; each edition (delimited by project-specific milestones) is clearly labelled and identified, and always coherent, consistent, and complete. As we have learned from the TEI’s incremental releases of the Guidelines, this is a far superior approach, as such releases are much easier to maintain and archive over the long term.

15 In addition, we do provide shortcuts through the build process for local testing of individual files. Our build can be parameterized by supplying one or two specific document ids as input, and in that case, the entire build runs for only those documents and the results are visible within a minute or so.

6. Conclusion

16 We will conclude by summarizing the intent and principles governing our build process:

- Everything that can be pre-fabricated should be pre-fabricated.
- Everything that could conceivably be useful should be created and included.
- Redundancy is beneficial; in fact it is elegant. If the same personography entry is replicated in fifty pages that mention that person, then good; any of those pages can now be used outside the context of the collection without loss.
- Patience is a virtue: let your build take a long time; let your releases be well-separated.

BIBLIOGRAPHY

Arneil, Stewart, and Martin Holmes. 2017. "Archiving form and function: preserving a 2003 digital project." DPASSH Conference 2017: Digital Preservation for Social Sciences and Humanities, Brighton, UK.

Holmes, Martin. 2017a. "Selecting Technologies for Long-Term Survival." SHARP Conference 2017: Technologies of the Book, Victoria, BC, Canada. https://github.com/projectEndings/Endings/raw/master/presentations/SHARP_2017/mdh_sharp_2017.pdf.

———. 2017b. "Whatever happened to interchange?" *Digital Scholarship in the Humanities*, Volume 32, Issue suppl_1, April 2017, Pages i63–i68. https://doi.org/10.1093/llc/fqw048.

Holmes, Martin, and Joseph Takeda. 2018. "Why do I need four search engines?" Japanese Association for Digital Humanities Conference, Tokyo, Japan. https://conf2018.jadh.org/files/Proceedings_JADH2018.pdf#page=58.

———. 2019. "Beyond Validation: Using Programmed Diagnostics to Learn About, Monitor, and Successfully Complete Your DH Project." In *Digital Scholarship in the Humanities*. Oxford University Press/EADH. http://dx.doi.org/10.1093/llc/fqz011.

JamStack: Modern web development architecture based on client-side JavaScript, reusable APIs, and prebuilt Markup. n.d. https://jamstack.org.

Kirschenbaum, Matthew. 2009. "Done: Finishing Projects in the Digital Humanities." *Digital Humanities Quarterly*, Volume 3, Issue 2. http://digitalhumanities.org:8081/dhq/vol/3/2/000037/000037.html.

Rockwell, Geoffrey, Shawn Day, Joyce Yu, and Maureen Engel. 2014. “Burying Dead Projects: Depositing the Globalization Compendium.” *Digital Humanities Quarterly*, Volume 8, Issue 2. http://digitalhumanities.org:8081/dhq/vol/8/2/000179/000179.html.

NOTES

[1] One of the perspectives Holmes brought to the *Endings* project was his familiarity with the static build process for the TEI Guidelines, for which we are primarily indebted to the late Sebastian Rahtz, a wise and clever man who realized all this a long time ago.

AUTHORS

MARTIN HOLMES

Programmer/Consultant, University of Victoria Humanities Computing and Media Centre

JOSEPH TAKEDA

MA Student, University of British Columbia